Systems Programming Concepts: Programming Assignment 7
(a.k.a. The Last One)

CS-202 **Winter 2010**

Assignment Weight: 2.0

In this programming assignment, you will demonstrate your ability to write client-server programs by writing both a client and a server program to play the contemporary game of "Rock-Paper-Scissors-Lizard-Spock".

Background

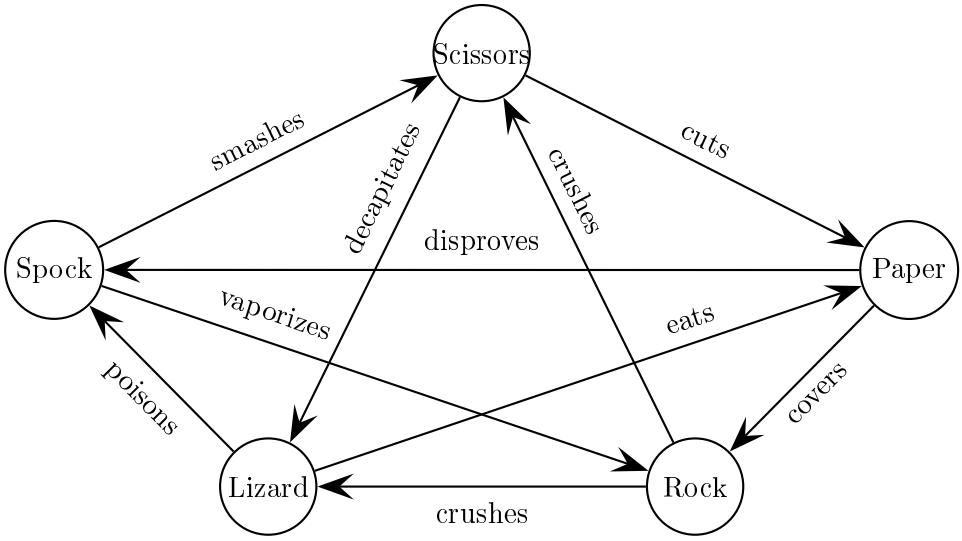
Most of you are familiar with the childhood game of "Rock-Paper-Scissors". A short explanation can be found on Wikipedia:

`http://en.wikipedia.org/wiki/Rock-paper-scissors`

Sam Kass invented an extension of this game, which includes two additional choices: "Lizard" and "Spock". The game plays exactly like the original game, under the following extended ruleset:

- Scissors cuts Paper and decapitates Lizard
- Paper covers Rock and disproves Spock
- Rock crushes Scissors and crushes Lizard
- Lizard eats Paper and poisons Spock
- Spock smashes Scissors and vaporizes Rock

A graph of these relationships appears on the next page. One can see by careful study that the extended game is fair, as is the original game; each choice beats two other choices and loses to two others. In addition, the presence of two additional choices reduces the chance that two players will make the same choice (which results in a tie).


Scissors
smashes
decapitates
crushes
cuts
Spock
disproves
Paper
vaporizes
eats
poisons
covers
Lizard
Rock
crushes

Server Requirements

You will create a file called "`server.c`" containing code for a server for games of Rock-Paper-Scissors-Lizard-Spock (hereafter RPSLS).

The server will begin by creating a socket on your machine (by default, on port 25000). It will wait until it receives two separate client requests, and then create a child process to manage a game of RPSLS between the two clients. (Thus, the server should be able to handle an indefinite number of games at the same time.)

Each game of RPSLS will begin by the server (child) process receiving from each client a "username" (an arbitrary string), which will then be sent to the other client, in order to identify the "opponent". The child process will then conduct a number of "turns". On each turn, the server will read a choice from each client, which will be one of the five answers noted above. The server will then send each client's choice to the other client, along with an indication of the result of the turn (win, loss, or tie).

The child process will continue conducting turns between the clients until one client wins at least two turns. (Note that, because of ties, this could take an indefinite number of turns.) At that point, the child will inform both clients of the final result and exit. If one client terminates before the end of the game, the game results in a win by forfeit for the other client.

It is up to you to define the communications protocol used in communication between the server and client. You may pass information between the server and clients in any manner you like — as long as, of course, your server and clients successfully communicate with one another.

Client Requirements

You will create a file called "`client.c`" containing code for a client for games of RPSLS, compatible with the server requirements outlined above.

The client's requirements can be deduced from the description of the server. The client will begin by prompting the user for the name of the machine hosting the game, as well as a "username" (string). It will then play its part in conducting the game by, on each turn, prompting the user for each choice in the game, exchanging information with the server, displaying the results, and exiting once the game is over.

Again, it is up to you to define how the user interacts with the client program. You may interact with the user, and the server, in any manner you like — as long as, of course, your server and clients successfully communicate with one another, and your client makes clear to the user how to use the client.

Submitting Your Program

Before 11:59:59 p.m., Tuesday, 23 March 2010 (11th Tuesday), you must send a MIME-encoded email message to `jhuggins@kettering.edu` containing all source code files for your program.

In addition, you must deliver to the instructor a printout of your program files at the start of the final exam.

Suggestions

1. I suggest that you begin your work on this project by carefully defining the protocol used between your client and your server. It may assist you immensely during debugging if you define your protocol in terms of strings, so that your clients and server can echo those strings to standard output while your program is running.

2. Note that for this program to work, a given process will have to both read and write to the same file descriptor produced by the socket. This is completely normal, and expected behavior for sockets. For example, see how the simple webserver from Chapter 12 both reads from the socket's file descriptor (to get the request from the client) and then writes to the socket's file description (to send back the requested data).

3. Obviously, this program will be much easier to debug with three separate terminals running: one for the server, and one for each of two clients.

4. Remember that users are notorious for not following directions; however you decide to have your client interact with the user, the client should be prepared for unexpected inputs from the user.